<u>In the claims:</u>

For the Examiner's convenience, all pending claims are presented below with changes shown in accordance with the mandatory amendment format.

1. (Currently Amended) A method comprising:

for a first thread, entering a processing queue for obtaining permission to enter a critical section of code;

determining if a second thread exists, the second thread executing the critical section of code concurrently with the first thread entering the processing queue; and

if the second thread exists, then determining if the second thread is executing the critical section;

if the second thread is executing the critical section, then testing for the second thread to complete until one of the following occurrences:

the second thread completes; and

a yielding count expires <u>and the first thread relinquishes its remaining quantum time to another thread in the processing queue</u>, wherein the yielding count is the amount of time the first thread is to remain in the processing queue.

2. (Original) The method of claim 1, additionally comprising if the yielding count expires before the second thread completes, then exiting the processing queue.

3. (Original) The method of claim 2 additionally comprising re-entering the processing queue after a period of time.

4. (Original) The method of claim 3, wherein the period of time is determined by an operating system scheduling algorithm.

5. (Original) The method of claim 1, additionally comprising if the second thread completes before the yielding count expires, then executing the first critical section of code.

6. (Original) The method of claim 1, additionally comprising if the second thread does not exist, then executing the first critical section of code.

7. (Original) The method of claim 1, wherein the yielding count is based on the number of threads contending to enter a corresponding critical section of code.

8. (Original) The method of claim 7, wherein the yielding count is based on twice the number of threads contending for the lock.

9. (Original) The method of claim 1, wherein the yielding count is based on the number of CPUs (central processing units).

10. (Original) The method of claim 1, wherein the critical section of code includes the same code in both the first and the second thread.

11. (Currently Amended) A method comprising:

for a first thread, entering a processing queue for obtaining a lock on a shared resource in a first critical section of code by checking the status of shared variables existing in a memory, the shared variables including a turn variable and a status flag;

determining if a second thread exists, the second thread executing a second critical section of code concurrently with the first thread entering the processing queue, the second critical section corresponding to the second thread; and

if the second thread exists, then testing for the second thread to relinquish the lock on the shared resource by testing the status flag, the testing to be performed until one of the following occurrences:

the second thread relinquishes the lock when the flag has been reset; and

a yielding count expires and the first thread relinquishes its remaining quantum time to another thread in the processing queue, wherein the yielding count is the amount of time the first thread is to remain in the processing queue.

12. (Original) The method of claim 11, wherein if the second thread relinquishes the lock before the yielding count expires, then obtaining the lock on the shared resource.

13. (Original) The method of claim 11, wherein if the yielding count expires before the second thread completes, then exiting the processing queue.

14. (Original) The method of claim 13, additionally comprising re-entering the processing queue after a determined amount of time.

15. (Currently Amended) A method comprising:

a. initializing shared variables, the shared variables including a turn variable, a first status flag, and a second status flag;

b. reading the shared variables into a memory;

c. entering a processing queue;

d. determining if a yield count has expired, wherein the yielding count is the amount of time a ~~the~~ first thread is to remain in the processing queue;

e. if the yield count has expired, then exiting the processing queue and the first thread relinquishes its remaining quantum time to another thread in the processing queue;

f. if the yield count has not expired, then for a contending process, determining if a concurrent process exists;

g. retrieving the second status flag and the turn variable from the memory, reading the <u>second</u> status flag into a first cache and reading the turn variable into the second cache to determine if the concurrent process is executing a critical section of code;

h. if the concurrent process is not executing a critical section of code, then entering the critical section of code, and upon completing the critical section of code, resetting the first status flag; and;

i. if the concurrent process is executing the critical section of code, then repeating d through i.

16. (Original) The method of claim 15, wherein the second cache is larger than the first cache.

17. (Original) The method of claim 15, wherein resetting the first status flag comprises retrieving a reset value from a register.

18. (Currently Amended) A machine-readable medium having stored thereon data representing sequences of instructions, the sequences of instructions which, when executed by a processor, cause the processor to perform operations comprising:

for a first thread, entering a processing queue for obtaining permission to enter a critical section of code;

determining if a second thread exists, the second thread executing the critical section of code concurrently with the first thread entering the processing queue; and

if the second thread exists, then determining if the second thread is executing the critical section;

if the second thread is executing the critical section, then testing for the second thread to complete until one of the following occurrences:

the second thread completes; and

a yielding count expires and the first thread relinquishes its remaining quantum time to another thread in the processing queue, wherein the yielding count is the amount of time the first thread is to remain in the processing queue.

19. (Previously Presented) The machine-readable medium of claim 18, additionally including data that causes the processor to perform operations comprising if the yielding count expires before the second thread completes, then exiting the processing queue.

20. (Previously Presented) The machine-readable medium of claim 18, additionally including data that causes the processor to perform operations comprising if the second thread does not exist, then executing the first critical section of code.

21. (Currently Amended) An apparatus comprising:

at least one processor;

a machine-readable medium having instructions encoded thereon, which when executed by the processor, are capable of directing the processor to perform operations comprising:

for a first thread, entering a processing queue for obtaining permission to enter a critical section of code;

determining if a second thread exists, the second thread executing the critical section of code concurrently with the first thread entering the processing queue; and if the second thread exists, then testing for the second thread to complete execution of the critical section of code until one of the following occurrences:

the second thread completes; and

a yielding count expires and the first thread relinquishes its remaining quantum time to another thread in the processing queue, wherein the yielding count is the amount of time the first thread is to remain in the processing queue.

22. (Previously Presented) The apparatus of claim 21, wherein the machine-readable medium additionally includes data that causes the processor to perform operations comprising if the yielding count expires before the second thread completes, then exiting the processing queue.

23. (Previously Presented) The apparatus of claim 21, wherein the machine-readable medium additionally includes data that causes the processor to perform operations comprising if the second thread does not exist, then executing the first critical section of code.

24. (Currently Amended) An apparatus comprising:

means for a first thread to enter a processing queue for obtaining permission to enter a critical section of code;

means to determine if a second thread exists, the second thread executing the critical section of code concurrently with the first thread entering the processing queue; and

if the second thread exists, then means to determine if the second thread is executing the critical section;

if the second thread is executing the critical section, then means to test for the second thread to complete until one of the following occurrences:

the second thread completes; and

a yielding count expires and the first thread relinquishes its remaining quantum time to another thread in the processing queue, wherein the yielding count is the amount of time the first thread is to remain in the processing queue.

25. (Previously Presented) The apparatus of claim 24, additionally comprising means for exiting the processing queue if the yielding count expires before the second thread completes.

26. (Previously Presented) The apparatus of claim 25 additionally comprising means for re-entering the processing queue after a period of time.

27. (Previously Presented) The apparatus of claim 26, wherein the period of time is determined by an operating system scheduling algorithm.

28. (Previously Presented) The apparatus of claim 24, additionally comprising means for executing the first critical section of code if the second thread completes before the yielding count expires.

29. (Previously Presented) The apparatus of claim 24, additionally comprising means for executing the first critical section of code if the second thread does not exist.

30. (Previously Presented) The apparatus of claim 24, wherein the yielding count is based on the number of threads contending to enter a corresponding critical section of code.